Maple and wave-front tracking : an experiment (Feasibility of Theoretical Arguments of Mathematical Analysis on Computer)

AUTHOR(S):

Yoshikawa, Atsushi

数理解析研究所講究録 1381 巻 2004 年 190-218

Maple and wave-front tracking — an experiment

Atsushi YOSHIKAWA
Kyushu University
Faculty of Mathmatics
yoshikaw@math.kyushu-u.ac.jp

1 Introduction

Consider the initial-value problem of a conservation law with a locally Lipschitz continuous flux $f(u)$

$$\frac{\partial}{\partial t}u(t,x)+\frac{\partial}{\partial x}\left(f(\,u(t,x)\,)\right)=0,\quad t>0,\ -\infty<x<+\infty, \tag{1}$$

$$u(0,x)=g(x). \tag{2}$$

It is known that the initial value problem (1)(2) has a unique *entropy* solution $u(t,x)$ if $g(x)$ is summable and of bounded variation on the real line. Moreover, for each $t>0$, $u(t,\cdot)$ is summable and of bounded variation (in x) on the line.

Discussions related to these facts are classical. In particular, the polygonal approximation due to Dafermos[2] establishes an algorithm which would eventually yield to the entropy solution of (1)(2). At least, when $g(x)$ is a step function with rational values and jump points, this is in fact true for those $f(u)$ which are piecewise linear with rational values and corner points . The key observation of Dafermos is that, for piecewise linear flux f, the related Riemann problem, that is, (1)(2) with $g(x)$ of two distinct constant values on the positive and negative half-lines, always yields a piecewise constant (entropy) solution. This allows an algorithmic handling of interactions of such solution-waves arising through localized Riemann problems. If the initial $g(x)$ is a finite step function, then the total number of interactions is estimated a priori by mathematical analysis. Thus, there must be a finite algorithm for an entropy solution in such a case.

Actually Dafermos has devised these piecewise linear fluxes as approximations to the original flux, and then sought for the solutions of thus obtained approximating problems. These approximating solutions are finally shown to converge to the entropy solution of the original problem. Note, however, effectivity of the last convergence is yet to be shown from the viewpoint of Computability Analysis (See, e.g., [5]).

The purpose of the present note is to give Maple[1] codes of this algorithm, and verify its validity for some cases of piecewise linear f and steplike g. In

[1]Maple is the registered trademark of Waterloo Maple, Inc.

particular, we include graphs of thus obtained entropy solutions and also their discontinuity lines.

2 Minimal prerequisites

To begin with, we recall that $f(u)$ is a locally Lipschitz continuous function (on the real line $\mathbb{R}$) if there is a constant $L_R > 0$ (called Lipschitz constant) for any $R > 0$ such that

$$|f(u) - f(v)| \leq L_R |u - v| \quad \text{for} \quad |u|, |v| \leq R.$$

In particular, any polynomial function is locally Lipscitz.

Let $f_N(u)$ be a piecewise linear approximation of the flux connecting the values of f at dyadic points $v_m = \dfrac{m}{2^N}$. Thus,

$$f_N(v) = \begin{cases} f(v), & v = v_m \\ (1-\theta)\, f(v_m) + \theta\, f(v_{m+1}), & v = (1-\theta)\, v_m + \theta\, v_{m+1} \end{cases} \tag{3}$$

$(0 < \theta < 1)$. It is not difficult to see that $f_N(u)$ is locally Lipschitz (with the corresponding Lipschitz constant $\leq L_{R+1/2^N}$).

Return to the equations (1)(2). Following Kruzhkov[3], we call a locally integrable function $u = u(t,x)$ which satisfies (2) an entropy solution of (1)(2) if

$$\int_0^{\infty} \int_{-\infty}^{+\infty} \{|u-k|\, \varphi_t + (f(u) - f(k))\, \mathrm{sgn}(u-k)\, \varphi_x\}\, dtdx \geq 0 \tag{4}$$

holds for any $k \in \mathbb{R}$ and for any φ. Here $\varphi = \varphi(t,x)$ is an arbitrary non-negative smooth function which vanishes for $t \leq$ or for large $|x| + |t|$. Observe that (4) is also valid if f is replaced by any piecewise linear approximation f_N

Then it is known that an entropy solution to the problem (1)(2) is uniquely determined for locally summable $g(x)$ ([3]).

Suppose $g_N(x)$ is a step function with values $\dfrac{q}{2^N}$. We then propose an algorithm which gives the entropy solution $u_N(t,x)$ of the conservation law with flux $f_N(u)$

$$\frac{\partial}{\partial t} u(t,x) + \frac{\partial}{\partial x} (f_N(\, u(t,x)\,)) = 0, \quad t > 0, \ -\infty < x < +\infty, \tag{5}$$

$$u(0,x) = g_N(x). \tag{6}$$

The algorithm is based on the mathematical analysis of C. Dafermos[2] (cf. Bressan [1]). We experiment this algorithm by the software Maple (§6).

3 Dafermos' key observations

Choose $g(x)$ of (2) as

$$g(x) = \begin{cases} u_-, & x < 0, \\ u_+, & x > 0 \end{cases} \tag{7}$$

where u_- and u_+ are distinct constants. Then the question of finding a solution satisfying (1)(2) is called *a Riemann problem.*

We now take dyadic rationals as $u_\pm$:

$$u_\pm = \frac{m_\pm}{2^N}, \quad m_-, m_+ \in \mathbb{Z},\ m_- \neq m_+. \tag{8}$$

Consider (1) with f replaced by its piecewise linear approximation. Then the entropy solution to the corresponding Reimann problem is explicitly given in the following way.

First we determine auxiliary functions $f_N^*(u; u_-, u_+)$ and $f_{N*}(u; u_-, u_+)$ from f_N, u_-, u_+.

Case 1. Suppose $u_- > u_+$. Let $f_{N*}(u; u_-, u_+)$ be the minimal *upper convex* function such that

$$f_{N*}(u; u_-, u_+) \geq f_N(u), \quad u_+ \leq u \leq u_-.$$

Case 2. Suppose $u_- < u_+$. Let $f_N^*(u; u_-, u_+)$ be the maximal *lower convex* function such that

$$f_N^*(u; u_-, u_+) \leq f_N(u), \quad u_- \leq u \leq u_+.$$

The functions $f_{N*}(u; u_-, u_+)$ and $f_N^*(u; u_-, u_+)$ are piecewise linear on the definition domain.

Thus, for Case 1, we have corner points $u_+ = u_{p*} < u_{p-1*} < \cdots < u_{1*} = u_-$ and values $c_{p-1*} > \cdots > c_1*$ such that

$$f_{N*}(u; u_-, u_+) = f_N(u_{k+1*}) + c_{k*}\frac{u - u_{k*}}{u_{k*} - u_{k+1*}}$$
$$u_{k+1*} \leq u < u_{k*}$$

for $k = 1, \cdots, p-1$. Note that u_{k*} are dyadic rationals $\frac{m_{k*}}{2^N}$ while

$$c_{k*} = \frac{f_N(u_{k+1*}) - f_N(u_{k*})}{u_{k+1*} - u_{k*}}$$

For Case 2, we have corner points $u_1^* = u_- < u_2^* < \cdots < u_q^* = u_+$ and values $c_1^* < c_2^* < \cdots < c_{q-1}^*$ such that

$$f_N^*(u; u_-, u_+) = f_N(u_k^*) + c_k^*\frac{u - u_k^*}{u_{k+1}^* - u_k^*}$$
$$u_k^* \leq u < u_{k+1}^*$$

for $k = 1, \cdots, q-1$. Here u_k^* are dyadic rationals $\frac{m_k^*}{2^N}$ while

$$c_k^* = \frac{f_N(u_{k+1}^*) - f_N(u_k^*)}{u_{k+1}^* - u_k^*}.$$

Now the corresponding entropy solution $R_N(t,x;u_-,u_+)$ of the Riemann problem reads as follows:

For Case 1, i.e., when $u_- > u_+$,

$$R_N(t,x;u_-,u_+) = \begin{cases} u_{1*}, & x < c_{1*}t \\ \cdots, & \cdots \\ u_{k*}, & c_{k-1*}t \le x < c_{k*}t \\ \cdots, & \cdots \\ u_{p*}, & c_{p-1*}t \le x \end{cases}$$

and for Case 2, i.e., when $u_- < u_+$,

$$R_N(t,x;u_-,u_+) = \begin{cases} u_1^*, & x < c_1^*t \\ \cdots, & \cdots \\ u_k^*, & c_{k-1}^*t \le x < c_k^*t \\ \cdots, & \cdots \\ u_q^*, & c_{q-1}^*t \le x \end{cases} .$$

What is important in these solutions is that they contain wave elements or triplets $(x-ct-d,\, v_\ell,\, v_r)$, consisting discontinuous lines and values on the both sides. Furthermore, all these wave elements are algorithmically determined, for f_N^* or f_{N*} can be calculated once $u_\pm$ are given. Because of the forms of wave elements, it is easy to handle their interactions. For instance, if we have two (adjacent) elements $w_1 = (x-c_1t-d_1, v_{1\ell}, v_{1r})$ and $w_2 = (x-c_2t-d_2, v_{2\ell}, v_{2r})$, where discontinuous lines $x-c_1t-d_1 = 0$ and $x-c_2t-d_2 = 0$ intersect at some future time and the values v_{r1} and $v_{2\ell}$ in between are common, $v_{r1} = v_{2\ell}$, then we encounter a Riemann problem at the intersection time with the initial data $v_{1\ell}$ and v_{2r}. These are basically incorporated in an algorithm which eventually generates an entropy solution for general step initial data.

4 How to use the codes

Now we have Maple codes which generates the entropy solution (See §6). Suppose

$$g_N(x) = \begin{cases} vlist_1\,, & x < xlist_1 \\ \cdots & \\ vlist_{i+1}\,, & xlist_i < x < xlist_{i+1} \\ \cdots & \\ vlist_{n+1}\,, & xlist_n < x \end{cases}, \tag{9}$$

where *xlist* and *ylist* are finite lists of (explicitly given) rationals:

$$xlist = [xlist_1,\, xlist_2,\, \cdots,\, xlist_n] \tag{10}$$

$$vlist = [vlist_1,\, vlist_2,\, \cdots,\, vlist_n, vlist_{n+1}] \tag{11}$$

and the components of *xlist* is strictly monotone increasing.

You may then wish to compute the entropy solution $u(t,x)$ for (5) (6) and its discontinuity lines or fronts. You activate the Maple codes in §6. Choose a polynomial with rational coefficients as the function f. Give also *xlist* and *vlist* explicitly. Choose a positive integer N. Then

```
> u:=entropy_solution:-zenbukai(xlist,vlist,N,f,M):
```

produces[2] the entropy solution $u(t,x)$ correctly (without any error) up to M interactions of fronts (§6.7). Actually the total number of interactions of fronts is finite[3]. Thus, for very large M, the entropy solution in the large can thus be computed.

As for the configuration of fronts, we compute

```
> hurenzokusen:=tuiseki:-zensujisen(t,xlist,vlist,N,f,M):
```

and we get all the fronts correctly up to M interactions (§6.8).

u and hurenzokusen are computed without errors for rational inputs, but when they are visualized, it is done via numerical evaluation with floating decimals.

5 various graphs

Let $f : u \to u^3$ or $f(u) = u^3$, $N = 2$ and

$$g_N(x) = g(x) = \begin{cases} 0, & x < 0 \\ 1, & 0 < x < 1 \\ 2, & 1 < x < 2 \\ 0, & 2 < x \end{cases}$$

or $xlist = [0,1,2]$ and $vlist = [0,1,2,0]$. Choose $M = 15$, which actually exceeds the total number of interactions in the present choice of f, N and g_N.

Figure1 and Figure2 show outputs of u and hurenzokusen.

6 The Maple codes

We have transcribed Dafermos' idea into the Maple codes given below.

The flux $f(u)$ is assumed to be a polynomial with integer coefficients. In the output above, the case $f(u) = u^3$ has been chosen.

The following Maple codes are divided into several modules ([4]), and actually employed for constructing entropy solutions. The codes are not intended to be most efficient. I apologize for omitting any explanation about the codes while using the terminologies strongly reflecting the Japanese language[4].

---

[2] u corresponds to the mapping $u : (t,x) \to u(t,x)$.

[3] Though it can be estimated in advance from f, N and g_N, we have not included any Maple codes for this purpose.

[4] Any interested reader is cordially invited to contact me (with perhaps "Subject" RIMS0903Maple).

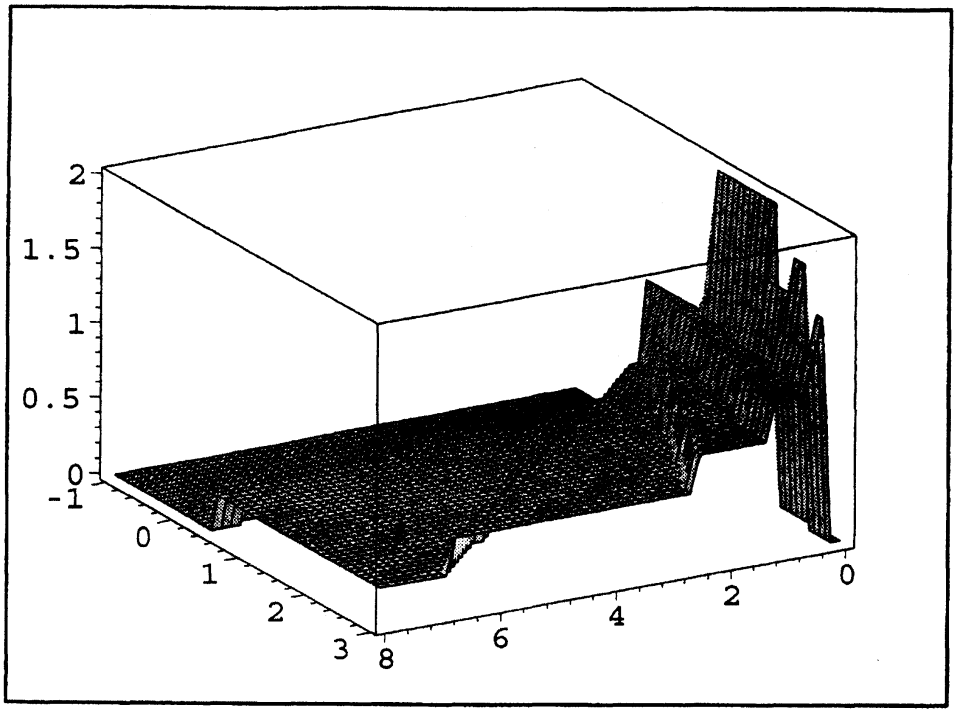

Figure 1: The graph of $u(t,x)$ for $0 < t < 8$, $-1 < x < 3$

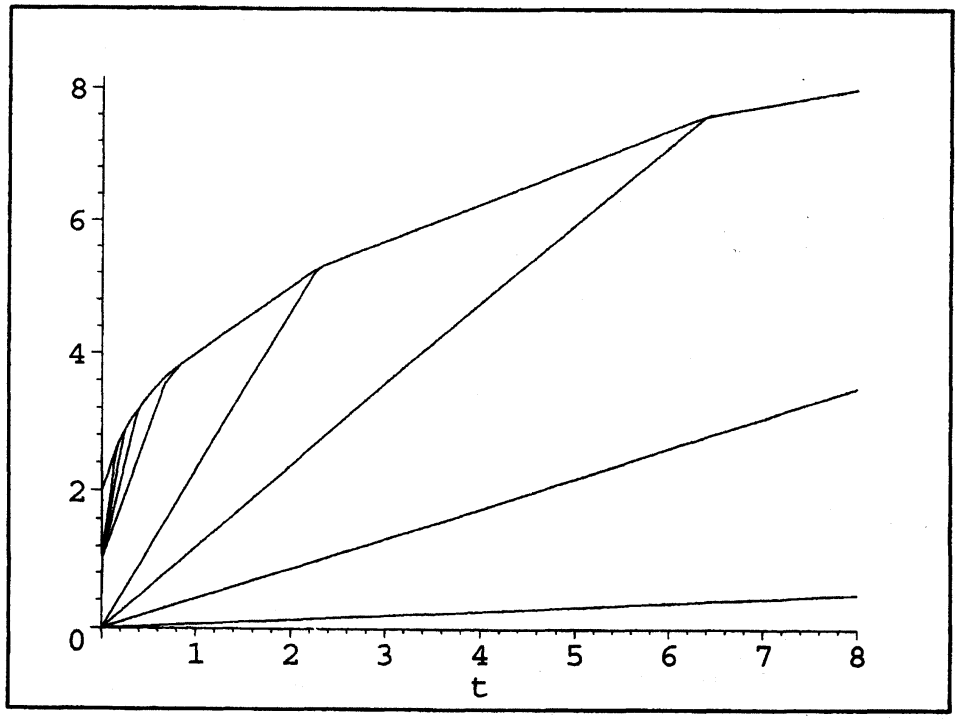

Figure 2: The configurations of fronts for $0 < t < 8$

6.1 module action

```
> action:=module()
> export dan, kaidan, oresen, sayoo;

> dan:=proc(b,v)
> local x:
> if b[1]>=b[2] then RETURN('wrong input') end if:
> x->piecewise(x<b[1],0,x>=b[1] and x<b[2], v, x>=b[2],0):
> end proc:

> kaidan:=proc(abscissae,values)
> local i,j,n,m,b,dd,dn,x:
> n:=nops(abscissae):
> m:=nops(values):
> if m<>n-1 then RETURN('bad data') end if:
> if abscissae<>sort(abscissae) then RETURN('confused data') end if:
> dn:=action:-dan:
> for i from 1 to n-1 do b[i]:=[abscissae[i],abscissae[i+1]] end do:
> for i from 1 to n-1 do dd[i]:=dn(b[i],values[i]) end do:
> x->sum(dd[j](x),j=1..n-1):
> end proc:

> oresen:=proc(abscissae,values)
> local k,kdn,t,x:
> kdn:=action:-kaidan:
> k:=t->kdn(abscissae,values)(t):
> x->int(k(t),t=-infinity..x):
> end proc:

> sayoo:=proc(abscissae,values)
> local f,orsn,t,x,y:
> orsn:=action:-oresen:
> f:=y->orsn(abscissae,values)(y):
> (t,x,y)->unapply((x-y)^2/(2*t)+f(y),(t,x,y)):
> end proc:

> end module:
```

6.2 module convexify

```
> convexify:=module()
> export gen, nawabari, katamuki, migikata, hidarikata,
> migisita, hidarisita, miginokori, hidarinokori,
> migiasi, hidariasi, ueetotu, sitaetotu,
> ueenototuka, sitaenototuka :
```

```
> gen:=proc(xpair,ypair)
> local x:
> if (nops(xpair)-2)^2+(nops(ypair)-2)^2>0
> then RETURN('convexify:-gen=input_error0') end if:
> if xpair[2]<=xpair[1] then RETURN('convexify:-gen=input_error1') end if:
> x->piecewise(x<xpair[1],0,x>xpair[1] and x<xpair[2],
> (xpair[2]-x)*ypair[1]/(xpair[2]-xpair[1])+
> (x-xpair[1])*ypair[2]/(xpair[2]-xpair[1]),
> x>xpair[2],0):
> end proc:

> nawabari:=proc(xlist,ylist)
> local i, j, n, gn, x:
> n:=nops(xlist):
> if n<2 then RETURN('convexify:-nawabari=input_error0') end if:
> if nops(ylist)<>n then RETURN('convexify:-nawabari=input_error1') end if:
> for i from 1 to n-1 do
> gn[i]:=gen([xlist[i],xlist[i+1]],[ylist[i],ylist[i+1]])
> end do:
> x->sum(gn[j](x),j=1..n-1):
> end proc:

> katamuki:=proc(xlist,ylist)
> local i, n, dir:
> n:=nops(xlist):
> if n<2 then RETURN('convexify:-katamuki=input_error0') end if:
> if nops(ylist)<>n then RETURN('convexify:-katamuki=input_error1') end if:
> for i from 1 to n-1 do
> dir[i]:=(ylist[i+1]-ylist[i])/(xlist[i+1]-xlist[i])
> end do:
> [seq(dir[i],i=1..n-1)]
> end proc:

> migikata:=proc(xlist,ylist)
> local i, n, dir:
> n:=nops(xlist):
> if n<2 then RETURN('convexify:-migikata=input_error0') end if:
> if nops(ylist)<>n then RETURN('convexify:-migikata=input_error1') end if:
> for i from 1 to n-1 do
> dir[i]:=(ylist[i+1]-ylist[1])/(xlist[i+1]-xlist[1])
> end do:
> [seq(dir[i],i=1..n-1)]
> end proc:

> hidarikata:=proc(xlist,ylist)
> local i, n, dir:
```

```
> n:=nops(xlist):
> if n<2 then RETURN('convexify:-hidarikata=input_error0') end if:
> if nops(ylist)<>n then RETURN('convexify:-hidarikata=input_error1') end if:
> for i from 1 to n-1 do
> dir[i]:=(ylist[n]-ylist[i])/(xlist[n]-xlist[i])
> end do:
> [seq(dir[i],i=1..n-1)]
> end proc:

> migisita:=proc(xlist,ylist)
> local a, b, i, n, ser, bango:
> a:=migikata(xlist,ylist):
> n:=nops(a):
> ser:=seq(a[i],i=1..n):
> b:=min(ser):
> bango:=:
> for i from 1 to n do
> if [ser][i]=b then bango:=bango union i end if
> end do:
> 1+min(seq(bango[i],i=1..nops(bango))):
> end proc;

> hidarisita:=proc(xlist,ylist)
> local  a, b, i, n, ser, bango:
> a:=hidarikata(xlist,ylist):
> n:=nops(a):
> ser:=seq(a[i],i=1..n):
> b:=min(ser):
> bango:=:
> for i from 1 to n do
> if [ser][i]=b then bango:=bango union i end if
> end do:
> max(seq(bango[i],i=1..nops(bango))):
> end proc;

> miginokori:=proc(xlist,ylist)
> local m, n, i:
> m:=migisita(xlist,ylist):
> n:=nops(xlist):
> [seq(xlist[i], i=m..n)], [seq(ylist[i],i=m..n)]
> end proc:

> hidarinokori:=proc(xlist,ylist)
> local m, i:
> m:=hidarisita(xlist,ylist):
> [seq(xlist[i],i=1..m)],[seq(ylist[i],i=1..m)]
> end proc:
```

```
> migiasi:=proc(xlist,ylist)
> local i, m, l, L, asi:
> L[0]:=xlist, ylist:
> m[0]:=0:
> l[0]:=1:
> asi:=l[0]:
> for i from 1 to nops(xlist) do
> m[i]:=migisita(L[i-1][1],L[i-1][2]):
> l[i]:=l[i-1]+m[i]-1:
> if l[i]<=nops(xlist) then
> asi:=asi union l[i]:
> L[i]:=miginokori(L[i-1][1],L[i-1][2])
> else i:=nops(xlist)+1
> end if:
> end do:
> asi:
> end proc:

> hidariasi:=proc(xlist,ylist)
> local i, m, l, L, asi:
> asi:=nops(xlist):
> L[0]:=xlist,ylist:
> m[0]:=nops(xlist):
> for i from 1 to nops(xlist)-1 do
> m[i]:=hidarisita(L[i-1][1],L[i-1][2]):
> if m[i]>=1 then
> asi:=asi union m[i]:
> L[i]:=hidarinokori(L[i-1][1],L[i-1][2])
> else i:=nops(xlist)
> end if
> end do:
> asi:
> end proc:

> sitaetotu:=proc(xlist,ylist)
> local i, j, asi, assi, Lx, Ly, X, Y, K, A:
> asi:=migiasi(xlist,ylist):
> Lx:=[seq(xlist[i],i=asi)]:
> Ly:=[seq(ylist[i],i=asi)]:
> K:=katamuki(Lx,Ly):
> if nops(K)=nops(seq(K[j],j=1..nops(K))) then
> X:=Lx:  Y:=Ly:
> else
> A:=:
> for i from 1 to nops(K)-1 do
> if K[i]=K[i+1] then
```

```
> A:=A union asi[i+1]
> end if
> end do:
> assi:=asi minus A:
> X:=[seq(xlist[i],i=assi)]:
> Y:=[seq(ylist[i],i=assi)]:
> end if:
> X, Y
> end proc:

> ueetotu:=proc(xlist,ylist)
> local i, j, asi, assi, Lx, Ly, X, Y, K, A:
> asi:=hidariasi(xlist,ylist):
> Lx:=[seq(xlist[i],i=asi)]:
> Ly:=[seq(ylist[i],i=asi)]:
> K:=katamuki(Lx,Ly):
> if nops(K)=nops(seq(K[j],j=1..nops(K))) then
> X:=Lx:  Y:=Ly:
> else
> A:=:
> for i from 1 to nops(K)-1 do
> if K[i]=K[i+1] then
> A:=A union asi[i+1]
> end if
> end do:
> assi:=asi minus A:
> X:=[seq(xlist[i],i=assi)]:
> Y:=[seq(ylist[i],i=assi)]:
> end if:
> X, Y
> end proc:

> ueenototuka:=proc(xlist,ylist)
> local x:
> x->nawabari(ueetotu(xlist,ylist))(x):
> end proc:

> sitaenototuka:=proc(xlist,ylist)
> local x:
> x->nawabari(sitaetotu(xlist,ylist))(x):
> end proc:

> end module:
```

6.3 module fcndata

```
> fcndata:=module()
```

```
> export fvalues, xdata, initdata, kinjidata, fcnkinji, totukinji, kinjitotuka:

> fvalues:=proc(xlist,f)
> local i, n:
> if xlist<>sort(xlist) then RETURN('fcndata:-fvalues=input_error0') end if:
> n:=nops(xlist):
> if nops(seq(xlist[i],i=1..n))<n then RETURN('fcndata:-fvalues=input_error1')
end if:
> map(f,xlist):
> end proc:

> xdata:=proc(m,n,N)
> local i:
> if m>=n then RETURN('fcndata:-xdata=input_error0') end if:
> [seq(i*2^(-N),i=m..n)]
> end proc:

> initdata:=proc(pair,N)
> local m, n:
> m:=2^N*pair[1]:
> n:=2^N*pair[2]:
> xdata(m,n,N):
> end proc:

> kinjidata:=proc(pair,N,f)
> local X, Y:
> X:=initdata(pair,N):
> Y:=fvalues(X,f):
> X,Y:
> end proc:

> fcnkinji:=proc(pair,N,f)
> local Z, x:
> Z:=kinjidata(pair,N,f):
> x->convexify:-nawabari(Z)(x):
> end proc:

> totukinji:=proc(pair,N,f,dir)
> local TK, KD:
> KD:=fcndata:-kinjidata(pair,N,f):
> if dir=ue then
> TK:=convexify:-ueetotu(KD):
> elif dir=sita then
> TK:=convexify:-sitaetotu(KD):
> else
> TK:=RETURN('totukinji=input_error')
> end if:
```

```
> TK:
> end proc;

> kinjitotuka:=proc(pair,N,f,dir)
> local t:
> unapply(convexify:-nawabari(totukinji(pair,N,f,dir))(t),t):
> end proc;

> end module:
```

6.4 module wavedata

```
> wavedata:=module()
> export sazanami, tatunami, namisuji, namisen, namisendata, majiwari, migiriemann,
hidaririemann, riemann, migiriemanndata, hidaririemanndata, riemanndata, riemannnamisen,
riemannnamisuji:

> sazanami:=proc(xlist,ylist)# output=[x,c,b,bb]
> local i, n, w, c:
> c:=convexify:-katamuki(xlist,ylist):
> n:=nops(c):
> for i from 1 to n do
> w[i]:=[xlist[i],c[i],ylist[i],ylist[i+1]]
> end do:
> seq(w[i],i=1..n):
> end proc;

> tatunami:=proc(xlist,ylist,t)# output a+c*t
> local i, n, w, c:
> c:=convexify:-katamuki(xlist,ylist):
> n:=nops(c):
> for i from 1 to n do
> w[i]:=unapply(xlist[i]+c[i]*t,t)
> end do:
> [seq(w[i](t),i=1..n)]:
> end proc;

> namisuji:=proc(datalist) # datalist=[a,c,b,bb]
> local t:
> unapply(datalist[1]+datalist[2]*t,t):
> end proc;

> namisen:=proc(datalist)# datalist=[a,c,b,bb]
> local t, nami:
> nami:=datalist[1]+datalist[2]*t:
> [unapply(nami,t),datalist[3],datalist[4]]:
> end proc;
```

```
> namisendata:=proc(datalist)#datalist=[t->a+c*t,b,bb]
> local t,u,a,c,b,bb:
> u:=unapply(datalist[1](t),t):
> a:=coeff(u(t),t,0):
> c:=coeff(u(t),t,1):
> b=datalist[2]:
> bb:=datalist[3]:
> [a,c,b,bb]:
> end proc;

> majiwari:=proc(namidata1,namidata2)# namidata=[t->a0+c*t,b1,b2]
> local t, nami1, nami2, T, a, b, bb:
> nami1:=namidata1[1]:
> nami2:=namidata2[1]:
> if nami1(0)=nami2(0)
> then RETURN('wavedata:-majiwari=input_error0')
> elif nami1(0)<nami2(0) then
> if namidata1[3]<>namidata2[2] then
> RETURN('wavedata:-majiwari=input_error1')
> elif coeff(nami1(t),t)<=coeff(nami2(t),t)
> then T:=infinity
> else T:=solve(nami1(t)-nami2(t),t):  a:=nami1(T):
> b:=namidata1[2]:  bb:=namidata2[3]
> end if
> else
> if namidata1[2]<>namidata2[3] then
> RETURN('wavedata:-majiwari=input_error2')
> elif coeff(nami2(t),t)<=coeff(nami1(t),t)
> then T:=infinity
> else T:=solve(nami1(t)-nami2(t),t):  a:=nami1(T):
> b:=namidata2[2]:  bb:=namidata1[3]
> end if
> end if:
> [T, a, [b,bb]]:
> end proc;

> migiriemann:=proc(Ulist,Flist)
> local s, Lu, Lf, dir, ddir, n, i:
> Lu:=convexify:-sitaetotu(Ulist,Flist)[1]:
> Lf:=convexify:-sitaetotu(Ulist,Flist)[2]:
> dir:=convexify:-katamuki(Lu,Lf):
> n:=nops(dir):
> ddir:=[-infinity,seq(dir[i],i=1..n),infinity]:
> s->action:-kaidan(ddir,Lu)(s):
> end proc;
```

```
> hidaririemann:=proc(Ulist,Flist)
> local s, Lu, Lf, dir, ddir, n, dddir, i, Luu:
> Lu:=convexify:-ueetotu(Ulist,Flist)[1]:
> Lf:=convexify:-ueetotu(Ulist,Flist)[2]:
> dir:=convexify:-katamuki(Lu,Lf):
> ddir:=sort(dir):
> n:=nops(ddir):
> dddir:=[-infinity,seq(ddir[i],i=1..n),infinity]:
> Luu:=ListTools[Reverse](Lu):
> s->action:-kaidan(dddir,Luu)(s):
> end proc;

> riemann:=proc(Pair,N,f)
> local pair, pseudoriemann,s:
> if Pair[1]=Pair[2] then RETURN('wavedata:-riemann=input_error0')
> elif Pair[1]>Pair[2] then pair:=[Pair[2],Pair[1]]:
> pseudoriemann:=s->hidaririemann(fcndata:-kinjidata(pair,N,f))(s):
> else pair:=Pair:
> pseudoriemann:=s->migiriemann(fcndata:-kinjidata(pair,N,f))(s):
> end if:
> unapply(pseudoriemann(s),s):
> end proc;

> migiriemanndata:=proc(Ulist,Flist,a)
> local s, Lu, Lf, dir, namidata, n, i:
> Lu:=convexify:-sitaetotu(Ulist,Flist)[1]:
> Lf:=convexify:-sitaetotu(Ulist,Flist)[2]:
> dir:=convexify:-katamuki(Lu,Lf):
> n:=nops(dir):
> for i from 1 to n do
> namidata[i]:=[a,dir[i],Lu[i],Lu[i+1]]
> end do:
> seq(namidata[i],i=1..n):
> end proc;

> hidaririemanndata:=proc(Ulist,Flist,a)
> local s, Lu, Lf, dir, ddir, n, namidata, i, Luu:
> Lu:=convexify:-ueetotu(Ulist,Flist)[1]:
> Lf:=convexify:-ueetotu(Ulist,Flist)[2]:
> dir:=convexify:-katamuki(Lu,Lf):
> ddir:=sort(dir):
> n:=nops(ddir):
> Luu:=ListTools[Reverse](Lu):
> for i from 1 to n do
> namidata[i]:=[a,ddir[i],Luu[i],Luu[i+1]]
> end do:
> seq(namidata[i],i=1..n):
```

```
> end proc;

> riemanndata:=proc(Pair,N,f,a) # output [a,c,b,bb]
> local pair, pseudoriemanndata,s:
> if Pair[1]=Pair[2] then RETURN('check data pair')
> elif Pair[1]>Pair[2] then pair:=[Pair[2],Pair[1]]:
> pseudoriemanndata:=hidaririemanndata(fcndata:-kinjidata(pair,N,f),a):
> else pair:=Pair:
> pseudoriemanndata:=migiriemanndata(fcndata:-kinjidata(pair,N,f),a):
> end if:
> pseudoriemanndata:
> end proc;

> riemannnamisen:=proc(Pair,N,f,a)
> local n, i, RD, RN :
> RD:=[riemanndata(Pair,N,f,a)]:
> n:=nops(RD):
> for i from 1 to n do
> RN[i]:=namisen(RD[i])
> end do:
> seq(RN[i],i=1..n):
> end proc;

> riemannnamisuji:=proc(Pair,N,f,a)
> local n, i, RD, RN :
> RD:=[riemanndata(Pair,N,f,a)]:
> n:=nops(RD):
> for i from 1 to n do
> RN[i]:=namisuji(RD[i])
> end do:
> seq(eval(RN[i]),i=1..n):
> end proc;

> end module:
```

6.5 module riemann

```
> riemann:=module()
> export tandata, migidata, hidaridata, tann, zendata, zendataretu, zenvdata,
namisuji, zennamisuji, namisujiplot, zennamisujiplot, zen, migihajidata, hidarihajidata,
migihajinamisuji, hidarihajinamisuji, migihajinamisen, hidarihajinamisen, tannamisen,
zennamisen;


> tandata:=proc(a,Pair,N,f)
> local pair, RD;
> if Pair[1]=Pair[2] then RETURN('riemann:-tandata=input_error0')
> elif Pair[2]<Pair[1] then pair:=[Pair[2],Pair[1]];
```

```
> RD:=hidaridata(a,fcndata:-kinjidata(pair,N,f))
> else pair:=Pair;
> RD:=migidata(a,fcndata:-kinjidata(pair,N,f))
> end if;
> RD;
> end proc;

> tann:=proc(a, Pair,N,f)
> local t, x, RD, n, i, j, RDD, TN:
> RD:=[tandata(a,Pair,N,f)]:
> n:=nops(RD):
> RDD[0]:=[a,-infinity,null,null]:RDD[n+1]:=[a,infinity,RD[n][4],null]:
> for i from 1 to n do RDD[i]:=RD[i] end do:
> for i from 1 to n+1 do
> TN[i]:=piecewise(x<RDD[i-1][2]*t+a,0,x>=RDD[i-1][2]*t+a and x<RDD[i][2]*t+a,RDD[i][3],
> x>RDD[i][2]*t+a,0)
> end do:
> unapply(sum(TN[j],j=1..n+1),(t,x)):
> end proc;

> zendata:=proc(xlist,vlist,N,f)
> local i, n, RD:
> if xlist<>sort(xlist) then RETURN('riemann:-zendata=input_error0') end if:
> n:=nops(xlist):
> if n+1<>nops(vlist) then RETURN('riemann:-zendata=input_error1') end if:
> for i from 1 to n do
> RD[i]:=[tandata(xlist[i],[vlist[i],vlist[i+1]],N,f)]
> end do:
> seq(RD[i],i=1..n):
> end proc;

> migidata:=proc (a, vlist, flist)
> local s, Lu, Lf, dir, namidata, n, i;
> Lu := convexify:-sitaetotu(vlist,flist)[1];
> Lf := convexify:-sitaetotu(vlist,flist)[2];
> dir := convexify:-katamuki(Lu,Lf);
> n := nops(dir);
> for i to n do
> namidata[i] := [a, dir[i], Lu[i], Lu[i+1]]
> end do;
> seq(namidata[i],i = 1 .. n)
> end proc;

> hidaridata:=proc(a,vlist,flist)
> local Lu, Lf, dir, ddir, n, Luu, i, namidata;
> Lu:=convexify:-ueetotu(vlist,flist)[1];
> Lf:=convexify:-ueetotu(vlist,flist)[2]:
```

```
> dir:=convexify:-katamuki(Lu,Lf);
> ddir:=sort(dir);
> n:=nops(ddir);
> Luu:=ListTools[Reverse](Lu);
> for i from 1 to n do
> namidata[i]:=[a,ddir[i],Luu[i],Luu[i+1]]
> end do;
> seq(namidata[i], i=1..n);
> end proc;

> migihajidata:=proc(a,Pair,N,f)
> local n:
> n:=nops([tandata(a,Pair,N,f)]):
> [tandata(a,Pair,N,f)][n]:
> end proc;

> hidarihajidata:=proc(a,Pair,N,f)
> [tandata(a,Pair,N,f)][1]:
> end proc;

> migihajinamisuji:=proc(a,Pair,N,f)
> wavedata:-namisuji(migihajidata(a,Pair,N,f)):
> end proc;

> hidarihajinamisuji:=proc(a,Pair,N,f)
> wavedata:-namisuji(hidarihajidata(a,Pair,N,f)):
> end proc;

> namisuji:=proc(a,Pair,N,f)
> local n, i, RD, RN :
> RD:=[tandata(a,Pair,N,f)]:
> n:=nops(RD):
> for i from 1 to n do
> RN[i]:=wavedata:-namisuji(RD[i])
> end do:
> seq(eval(RN[i]),i=1..n):
> end proc;

> zennamisuji:=proc(xlist,vlist,N,f)
> local n, i, NS, t:
> if xlist<>sort(xlist) then RETURN(`riemann:-namisuji=input_error0`) end if:
> n:=nops(xlist):
> if n+1<>nops(vlist) then RETURN(`riemann:-namisuji=input_erro1`) end if:
> for i from 1 to n do
> NS[i]:=namisuji(xlist[i],[vlist[i],vlist[i+1]],N,f) end do:
> seq(eval(NS[i]),i=1..n):
> end proc;
```

```
> migihajinamisen:=proc(a,Pair,N,f)
> wavedata:-namisen(migihajidata(a,Pair,N,f)):
> end proc;

> hidarihajinamisen:=proc(a,Pair,N,f)
> wavedata:-namisen(hidarihajidata(a,Pair,N,f)):
> end proc;

> tannamisen:=proc(a,vpair,N,f)
> wavedata:-riemannnamisen(vpair,N,f,a):
> end proc;

> zennamisen:=proc(xlist,vlist,N,f)
> local n, i, NS, t:
> if xlist<>sort(xlist) then RETURN('riemann:-zennamisen=input_error0') end if:
> n:=nops(xlist):
> if n+1<>nops(vlist) then RETURN('riemann:-zennamisen=input_error1') end if:
> for i from 1 to n do
> NS[i]:=tannamisen(xlist[i],[vlist[i],vlist[i+1]],N,f) end do:
> seq(eval(NS[i]),i=1..n):
> end proc;

> zendataretu:=proc(xlist,vlist,N,f,k)
> local RZD, ZD, n, m, i, j:
> RZD:=[zendata(xlist,vlist,N,f)]:
> n:=nops(RZD):
> for i from 1 to n do
> m[i]:=nops(RZD[i])
> end do:
> ZD:=seq(seq(RZD[i][j],j=1..m[i]),i=1..n):
> end proc;

> zenvdata:=proc(xlist,vlist,N,f)
> local n, i, ZDR:
> ZDR:=zendataretu(xlist,vlist,N,f):
> n:=nops([ZDR]):
> [ZDR[1][3],seq(ZDR[i][4],i=1..n)]:
> end proc;

> zen:=proc(xlist,vlist,N,f)
> local t,x,i,j,n,ZS,ZV,TN:
> ZS:=[zennamisuji(xlist,vlist,N,f)]:
> ZV:=zenvdata(xlist,vlist,N,f):
> n:=nops(ZS):
> for i from 1 to n-1 do
> TN[i]:=piecewise(x<ZS[i](t),0, x>ZS[i](t) and x<ZS[i+1](t),ZV[i+1],x>ZS[i+1](t),0)
```

```
> end do:
> unapply(sum(TN[j],j=1..n-1),(t,x)):
> end proc;

> namisujiplot:=proc(a,Pair,N,f,Tpair)
> local n, i, t, RN, RNP:
> if Tpair[1]>=Tpair[2] then
> RETURN('riemann:-namisujiplot=input_error0')
> end if:
> RN:=[namisuji(a,Pair,N,f)]:
> n:=nops(RN):
> for i from 1 to n do
> RNP[i]:=plot(RN[i](t),t=Tpair[1]..Tpair[2],color=black,xtickmarks=3,ytickmarks=3)
> end do:
> plots[display](seq(RNP[i],i=1..n)):
> end proc;


> zennamisujiplot:=proc(xlist,vlist,N,f,Tpair)
> local ZR, t, n, i, d:
> ZR:=zennamisuji(xlist,vlist,N,f):
> n:=nops([ZR]):
> for i from 1 to n do
> d[i]:=plot(ZR[i](t),t=Tpair[1]..Tpair[2],color=black,xtickmarks=2,thickness=2)
> end do:
> plots[display](seq(d[i],i=1..n)):
> end proc:

> end module:
```

6.6 module kansho

```
> kansho:=module()
> export jikokudata, jikoku, jikokuindices, zendatalist, kanshodata, gattai, datalist,
sinxdata, sinvdata, jikokuretu, kurikaesikanshodataretu;

> jikokudata:=proc(xlist,vlist,N,f)
> local n,i,R,L,T,Tm, S:
> if xlist<>sort(xlist) then RETURN('kansho:-jikokudata=input_error0') end if:
> n:=nops(xlist):
> if n+1<>nops(vlist) then RETURN('kansho:-jikokudata=input_error1') end if:
> for i from 1 to n do
> R[i]:=riemann:-migihajinamisen(xlist[i],[vlist[i],vlist[i+1]],N,f)
> end do:
> for i from 1 to n do L[i]:=riemann:-hidarihajinamisen(xlist[i],[vlist[i],vlist[i+1]],N,f)
> end do:
```

```
> for i from 1 to n-1 do
> T[i]:=wavedata:-majiwari(R[i],L[i+1])[1]
> end do:
> Tm:=min(seq(T[i],i=1..n-1)):
> S:=:
> for i from 1 to n-1 do
> if T[i]=Tm then S:=S union i
> end if
> end do:
> seq(wavedata:-majiwari(R[i],L[i+1]),i=S):
> end proc;

> jikoku:=proc(xlist,vlist,N,f)
> [jikokudata(xlist,vlist,N,f)][1][1]:
> end proc;

> jikokuindices:=proc(xlist,vlist,N,f)
> local n,i,R,L,T,Tm, S:
> if xlist<>sort(xlist) then RETURN('kansho:-jikokuindices=input_error0') end
if:
> n:=nops(xlist):
> if n+1<>nops(vlist) then RETURN('kansho:-jikokuindices=input_error1') end if:
> for i from 1 to n do
> R[i]:=riemann:-migihajinamisen(xlist[i],[vlist[i],vlist[i+1]],N,f)
> end do:
> for i from 1 to n do L[i]:=riemann:-hidarihajinamisen(xlist[i],[vlist[i],vlist[i+1]],N,f)
> end do:
> for i from 1 to n-1 do
> T[i]:=wavedata:-majiwari(R[i],L[i+1])[1]
> end do:
> Tm:=min(seq(T[i],i=1..n-1)):
> if Tm=infinity then S:=
> else
> S:=:
> for i from 1 to n-1 do
> if T[i]=Tm then S:=S union i
> end if
> end do:
> end if:
> S:
> end proc;

> zendatalist:=proc(xlist,vlist,N,f)
> local ZR, n, m, i, j, b, T, ZK:
> ZR:=riemann:-zendata(xlist,vlist,N,f):
> n:=nops(xlist):  # nops(xlist)=nops([ZR])
> for i from 1 to n do m[i]:=nops(ZR[i]) end do:
```

```
> T:=jikoku(xlist,vlist,N,f):
> for i from 1 to n do
> for j from 1 to m[i] do
> ZK[i][j]:=[ZR[i][j][1]+T*ZR[i][j][2],ZR[i][j][3],ZR[i][j][4]]:
> end do:
> end do:
> for i from 1 to n do
> b[i]:=[seq(ZK[i][j],j=1..m[i])]:
> end do:
> seq(b[i],i=1..n)
> end proc;

> kanshodata:=proc(xlist,vlist,N,f)
> local ZKD, KJI, n, m, i, j, KD:
> ZKD:=zendatalist(xlist,vlist,N,f):
> KJI:=jikokuindices(xlist,vlist,N,f):
> n:=nops(xlist):  #nops(xlist)=nops([ZKD]):
> for i from 1 to n do m[i]:=nops(ZKD[i]) end do:
> [seq([ZKD[i][m[i]],ZKD[i+1][i]],i=KJI)]:
> end proc;

> gattai:=proc(triplet1,triplet2)
> local triplet:
> if [triplet1[1],triplet1[3]]<>[triplet2[1],triplet2[2]]
> then RETURN('kansho:-gattai=input_error') end if:
> if triplet1[2]<>triplet2[3] then triplet:=[triplet1[1],triplet1[2],triplet2[3]]
> else triplet:=NULL
> end if:
> triplet:
> end proc;

> datalist:=proc(xlist,vlist,N,f)
> local ZDL, DI, n, m, i, j, KD, KDD:
> ZDL:=zendatalist(xlist,vlist,N,f):
> n:=nops(xlist):
> DI:=jikokuindices(xlist,vlist,N,f):
> for i from 1 to n do if i in DI then m[i]:=nops(ZDL[i])-1
> else m[i]:=nops(ZDL[i])
> end if
> end do:
> for i from 1 to n do
> for j from 1 to m[i] do KD[i][j]:=ZDL[i][j]
> end do
> end do:
> for i from 1 to n do if i-1 in DI then
> KDD[i][1]:=gattai(ZDL[i-1][m[i-1]+1],ZDL[i][1]):
> for j from 2 to m[i] do KDD[i][j]:=ZDL[i][j] end do
```

```
> else
> for j from 1 to m[i] do KDD[i][j]:=ZDL[i][j] end do
> end if
> end do:
> seq(seq(KDD[i][j],j=1..m[i]),i=1..n):
> end proc;

> sinxdata:=proc(xlist,vlist,N,f)
> local n, DL:
> DL:=datalist(xlist,vlist,N,f):
> n:=nops([DL]):
> [seq(DL[i][1],i=1..n)]:
> end proc;

> sinvdata:=proc(xlist,vlist,N,f)
> local n, i, DL, VL:
> DL:=datalist(xlist,vlist,N,f):
> n:=nops([DL]):
> for i from 1 to n do VL[i]:=DL[i][2] end do:
> VL[n+1]:=DL[n][3]:
> [seq(VL[i],i=1..n+1)]
> end proc;

> jikokuretu:=proc(xlist,vlist,N,f,M)
> local i, sinxlist, sinvlist, JKK:
> sinxlist[1]:=xlist:sinvlist[1]:=vlist:
> JKK[1]:=jikoku(sinxlist[1],sinvlist[1],N,f):
> for i from 2 to M do
> sinxlist[i]:=sinxdata(sinxlist[i-1],sinvlist[i-1],N,f):
> sinvlist[i]:=sinvdata(sinxlist[i-1],sinvlist[i-1],N,f):
> JKK[i]:=jikoku(sinxlist[i],sinvlist[i],N,f):
> end do:
> seq(JKK[i],i=1..M):
> end proc;

> kurikaesikanshodataretu:=proc(xlist,vlist,N,f,M)
> local i, j, sinxlist, sinvlist, JKK, n:
> sinxlist[1]:=xlist:sinvlist[1]:=vlist:
> JKK[1]:=jikoku(sinxlist[1],sinvlist[1],N,f):n:=1:
> for i from 1 to M do if type(JKK[i],rational)=true then
> sinxlist[i+1]:=sinxdata(sinxlist[i],sinvlist[i],N,f):
> sinvlist[i+1]:=sinvdata(sinxlist[i],sinvlist[i],N,f):
> JKK[i+1]:=jikoku(sinxlist[i],sinvlist[i],N,f):n:=i:
> end if:
> end do:
> [seq(JKK[i],i=2..n)],[seq(sinxlist[i],i=1..n)],[seq(sinvlist[i],i=1..n)]:
> end proc;
```

```
> end module:
```

6.7 module entropy_solution

```
> entropy_solution:=module()
> export bubunkai, zenbukai;

> bubunkai:=proc(xlist,vlist,N,f,M)
> local i, n, KKD, SOL, t, x:
> KKD:=kansho:-kurikaesikanshodataretu(xlist,vlist,N,f,M):
> n:=nops(KKD[1]):
> for i from 1 to n+1 do
> SOL[i]:=unapply(riemann:-zen(KKD[2][i],KKD[3][i],N,f),(t,x)):
> end do:
> seq(SOL[i](t,x),i=1..n+1):
> end proc;

> zenbukai:=proc(xlist,vlist,N,f,M)
> local n, i, j, k,KKD, BBK, T, SOL, t, x:
> KKD:=kansho:-kurikaesikanshodataretu(xlist,vlist,N,f,M):
> n:=nops(KKD[1]):
> for j from 1 to n do
> T[j]:=sum(KKD[1][i],i=1..j)
> end do:
> BBK:=bubunkai(xlist,vlist,N,f,M):
> SOL[1]:=unapply(
> piecewise(t<T[1],BBK[1](t,x),t>T[1],0),(t,x)):
> for j from 2 to n do
> SOL[j]:=unapply(
> piecewise(t<T[j-1],0,t>T[j-1] and t<T[j],BBK[j](t-T[j-1],x),t>T[j],0),
> (t,x))
> end do:
> SOL[n+1]:=unapply(
> piecewise(t<T[n],0,t>T[n],BBK[n+1](t-T[n],x)),(t,x)):
> unapply(sum(SOL[k](t,x),k=1..n+1),(t,x)):
> end proc;



> end module:
```

6.8 module tuiseki

```
> tuiseki:=module()
> export husibusi, zenhusibusi, kusari, tuinarabe2, tuinarabe3, juzutunagitui,
juzutunagi, zenjuzutunagi, sujimejun, bubunsuji, sujisen, sujisenplot, zensujisen:

> husibusi:=proc(xlist,vlist,N,f)
> local n, i, j, k, ZD, KJI, HSM, HSM1, hb:
> n:=nops(xlist):
> ZD:=riemann:-zendata(xlist,vlist,N,f):
> KJI:=kansho:-jikokuindices(xlist,vlist,N,f):
> HSM[0]:=0:HSM1[0]:=0:
> for i from 1 to n do if i in KJI then
> HSM[i]:=nops(ZD[i])-1
> else
> HSM[i]:=nops(ZD[i])
> end if
> end do:
> for i from 1 to n do
> HSM1[i]:=HSM1[i-1]+HSM[i]
> end do:
> for i from 1 to n do
> for j from 1 to nops(ZD[i]) do
> hb[i][j]:=HSM1[i-1]+j
> end do:
> end do:
> seq(seq([i,hb[i][j]],j=1..nops(ZD[i])),i=1..n):
> end proc;

> zenhusibusi:=proc(xlist,vlist,N,f,M)
> local KKD, n, i, HB:
> KKD:=kansho:-kurikaesikanshodataretu(xlist,vlist,N,f,M):
> n:=nops(KKD[2]):
> for i from 1 to n do
> HB[i]:=[husibusi(KKD[2][i],KKD[3][i],N,f)]
> end do:
> seq(HB[i],i=1..n):
> end proc;

> kusari:=proc(pairlist)
> local n, i, j, S1, S2, Output :
> n:=nops(pairlist):
> S1:=[seq(pairlist[i][2],i=1..n-1)]:  S2:=[seq(pairlist[i+1][1],i=1..n-1)]:
> if S1=S2 then Output:=pairlist
> else Output:=NULL
> end if:
> Output:
> end proc;
```

```
> tuinarabe2:=proc(pairset1,pairset2)
> local a, b:
> [seq(seq(kusari([a,b]),a=pairset1),b=pairset2)]:
> end proc;

> tuinarabe3:=proc(pairset1,pairset2,pairset3)
> local a, b, c:
> [seq(seq(seq(kusari([a,b,c]),a=pairset1),b=pairset2),c=pairset3)]:
> end proc;

> juzutunagitui:=proc(pairlist1,pairlist2)
> local PL, n1, n2, s1, s2, i, j, n:
> n1:=nops(pairlist1):
> n2:=nops(pairlist2):
> n:=nops(pairlist1[1]):
> s1:=seq(nops(pairlist1[i]),i=1..n1):
> s2:=seq(nops(pairlist2[j]),j=1..n2):
> if nops(s1) union nops(s2) <>1 then
> RETURN('juzutunagitui=input_error')
> end if:
> PL:=:
> for i from 1 to n1 do
> for j from 1 to n2 do
> if pairlist1[i][n][2]=pairlist2[j][1][1]
> then PL:=PL union [i,j]
> end if
> end do
> end do:
> PL:
> end proc;

> juzutunagi:=proc(pairlist1,pairlist2)
> local JZ, n1, n2, n, m, i, j, k, l, JT, p, q:
> JZ:=juzutunagitui(pairlist1,pairlist2):
> n1:=nops(pairlist1):  n:=nops(pairlist1[1]):
> n2:=nops(pairlist2):  m:=nops(pairlist2[1]):
> JT:=:
> for i from 1 to n1 do
> for j from 1 to n2 do
> if [i,j] in JZ then
> JT:=JT union [seq(pairlist1[i][k],k=1..n),seq(pairlist2[j][l],l=1..m)]
> end if
> end do
> end do:
> p:=nops(JT):
> [seq(JT[q],q=1..p)]:
> end proc;
```

```
> zenjuzutunagi:=proc(xlist,vlist,N,f,M)
> local ZHB, ZHBS, n, i, j, q, r, TN, a, b, c, JT:
> ZHB:=zenhusibusi(xlist,vlist,N,f,M):
> n:=nops([ZHB]):
> for i from 1 to n do
> ZHBS[i]:=convert(ZHB[i],set)
> end do:
> r:=irem(n,3,'q'):
> if q=0 then if r=1 then TN[q+1]:=[seq([ZHB[j]],j=1..nops(ZHB))]
> elif r=2 then  TN[q+1]:=tuinarabe2(ZHBS[1],ZHBS[2])
> end if
> else
> for j from 1 to q do
> TN[j]:=tuinarabe3(ZHBS[3*j-2],ZHBS[3*j-1],ZHBS[3*j])
> end do:
> if r=2 then TN[q+1]:=tuinarabe2(ZHBS[n-1],ZHBS[n])
> elif r=1 then TN[q+1]:=[seq([ZHB[n][j]],j=1..nops(ZHB[n]))]
> else TN[q+1]:=NULL
> end if:end if;
> if q=0 then JT[q+1]:=TN[q+1]
> else JT[1]:=TN[1]:
> for i from 1 to q-1 do
> JT[i+1]:=juzutunagi(JT[i],TN[i+1])
> end do:
> if r=0 then JT[q+1]:=JT[q]
> else
> JT[q+1]:=juzutunagi(JT[q],TN[q+1])
> end if:
> end if:
> JT[q+1]:
> end proc;

> sujimejun:=proc(pair,xlist,vlist,N,f)
> local HB, n, p, q, i, j, m, mm, S:
> HB:=husibusi(xlist,vlist,N,f):
> n:=nops([HB]):
> if nops(HB minus pair)=n then RETURN('sujimejun=input_error') end if:
> p:=pair[1]:
> q:=pair[2]:
> if p=1 then m:=q
> else
> S:=:
> for i from 1 to n do if [p,i] in HB then S:=S union i end if end do:
> mm:=min(seq(j,j=S)):
> m:=q-mm+1:
> end if:
```

```
> [p,m]:
> end proc;

> bubunsuji:=proc(pair,Tpair,xlist,vlist,N,f)# Tpair
> local SJ, t, a, b, m,NS, BBS:
> SJ:=sujimejun(pair,xlist,vlist,N,f):
> a:=xlist[SJ[1]]: b:=[vlist[SJ[1]],vlist[SJ[1]+1]]:
> m:=SJ[2]:
> NS:=unapply([riemann:-namisuji(a,b,N,f)][m](t),t):
> BBS:=piecewise(t<Tpair[1],0,t>Tpair[1] and t<Tpair[2], NS(t), t>Tpair[2],0):
> unapply(BBS,t):
> end proc;

> sujisen:=proc(pairlist,xlist,vlist,N,f,M)
> local M1, KKD, T, S, i, j, k, n, t, bbs:
> KKD:=kansho:-kurikaesikanshodataretu(xlist,vlist,N,f,M):
> M1:=nops(KKD[1]):
> T:=[0,seq(KKD[1][i],i=1..M1),infinity]:
> S:=[seq(sum(T[j],j=1..i),i=1..M1+2)]:
> n:=nops(pairlist):
> for k from 1 to n do
> bbs[k]:=bubunsuji(pairlist[k],[T[1],T[k+1]],KKD[2][k],KKD[3][k],N,f):
> end do:
> unapply(sum(bbs[i](t-S[i]),i=1..n),t):
> end proc;

> sujisenplot:=proc(pairlist,xlist,vlist,N,f,M)
> local sjs, t, n, KKD, TT, i, j:
> KKD:=kansho:-kurikaesikanshodataretu(xlist,vlist,N,f,M):
> n:=nops(pairs):
> sjs:=sujisen(pairlist,xlist,vlist,N,f,M):
> TT:=sum(KKD[1][i],i=1..n):
> plot(sjs(t),t=0..TT,color=black):
> end proc;

> zensujisen:=proc(t,xlist,vlist,N,f,M)
> local PL, n, sjs, i:
> PL:=zenjuzutunagi(xlist,vlist,N,f,M):
> n:=nops(PL):
> for i from 1 to n do
> sjs[i]:=sujisen(PL[i],xlist,vlist,N,f,M)(t)
> end do:
> [seq(sjs[i],i=1..n)]:
> end proc;

> end module:
```

References

[1] Alberto Bressan. *Hyperbolic systems of conservation laws.* Oxford University Press, 2000.

[2] C. Dafermos. Polygonal approximation of solutions of the initial value problems for a conservation law. J. Math. Anal. Appl. **38** (1972) 33–41.

[3] S. Kruzhkov. First order quasilinear equations with several variables. Math. USSR Sb. **10** (1970) 217–243.

[4] M. B. Monagan & al. *Maple 8 Advanced Programming Guide.* Waterloo Maple Inc., 2002.

[5] Marian B. Pour-El & J. Ian Richards. *Computability in analysis and physics.* Springer-Verlag, 1989.